Linear-Time Enumeration of Maximal $k$-edge-connected Subgraphs in Large Networks by Random Contraction

Takuya Akiba *(U Tokyo)*
Yoichi Iwata *(U Tokyo)*
Yuichi Yoshida *(NII&PFI)*

Software available: [http://git.io/mkecs](http://git.io/mkecs)
Cohesive Subgraphs
One of the most popular models:

$k$-core

- The maximal subgraph such that
- all vertices in it have degree at least $k$
Applications of $k$-core

Network Analysis
- Influencer analysis [KGH+10]
- Self-similarity, connectivity and hierarchy [AHDBV06, AhBV08]
- Analyzing cooperation in networks [CHK+07]

Fingerprinting & Visualization
- Protein interaction [Altaf-Ul-Amin+06]

Vaccination & Crime Prevention
- Vaccination [Kitsak+10]
- Detecting financial crime [DLMP11]
Serious Problem of $k$-core

The $k$-core model says:

This is one 17-core since all the vertices have degree $\geq 17$

(a real subgraph from a co-author network)
Serious Problem of $k$-core

..., but, there seem to be three cohesive subgraphs

The $k$-core model cannot separate them 😞
The issue

• $k$-cores are sometimes not well connected

Recent solution [Zhou+, EDBT’12]

• Maximal $k$-Edge-Connected Subgraphs
• Use MkECSs instead of $k$-cores
New Model: $M_k$ECS

They are three $M_k$ECSs ($k=17$) ☺

(a real subgraph from a co-author network)
The existing algorithm [Zhou+, EDBT’12] is too slow!

• We address this issue

• We propose a much faster algorithm

• MkECS is now practical for large networks
Proposed Method
**<Definition>** Graph $G$ is $k$-edge-connected

$\iff$ remains connected after removing any $k - 1$ edges

$\iff$ Size of minimum cut $\geq k$

**<Problem>** Decompose graph $G$ into maximal subgraphs that are $k$-edge-connected. (The decomposition is unique.)
Find cuts with size $< k$ and decompose by them

$k = 3$

Common to the previous and proposed method
Algorithm Overview

One Iteration

Overall Algorithm

while $G \neq$ empty
1. if $\exists u \in V$ s.t. $\deg(u) < k$
   • Output and remove $u$
2. else
   • Contract a random edge

for sufficient number of times
• Apply above algo. for subgraphs
Outline

① High-level Description

② Implementation & Complexity

③ Number of iterations

Outline

One iteration

while $G \neq \text{empty}$

1. if $\exists u \in V$ s.t. $\deg u < k$
   • Output and remove $u$

2. else
   • Contract a random edge

for sufficient number of times

• Apply above algo. for subgraphs
Outline

1 High-level Description

2 Implementation & Complexity

3 Number of iterations

One iteration

while $G \neq$ empty
1. if $\exists u \in V$ s.t. $\deg(u) < k$
   • Output and remove $u$
2. else
   • Contract a random edge

for sufficient number of times
• Apply above algo. for subgraphs

Iterations
One Iteration \((k = 3)\)

while \(G \neq \text{empty}\)

1. if \(\exists u \in V\) s.t. \(\deg(u) < k\)
   - Output and remove \(u\)

2. else
   - Contract a random edge
One Iteration \((k = 3)\)

\[
\text{while } G \neq \text{ empty}
\]

1. \textbf{if } \exists u \in V \text{ s.t. } \deg(u) < k
   \begin{itemize}
   \item Output and remove \(u\)
   \end{itemize}

2. \textbf{else}
   \begin{itemize}
   \item Contract a random edge
   \end{itemize}
**One Iteration** \((k = 3)\)

\[
\textbf{while } G \neq \text{ empty} \\
1. \quad \textbf{if } \exists u \in V \text{ s.t. } \text{deg}(u) < k \\
   \quad \quad \cdot \text{Output and remove } u \\
2. \quad \textbf{else} \\
   \quad \quad \cdot \text{Contract a random edge}
\]
One Iteration \((k = 3)\)

while \(G \neq \) empty

1. if \(\exists u \in V\) s.t. \(\text{deg}(u) < k\)
   - Output and remove \(u\)

2. else
   - Contract a random edge
One Iteration ($k = 3$)

while $G \neq$ empty

1. if $\exists u \in V$ s.t. $\deg(u) < k$
   - Output and remove $u$

2. else
   - Contract a random edge
One Iteration \((k = 3)\)

\[
\text{while } G \neq \text{ empty}
\]

1. \(\text{if } \exists u \in V \text{ s.t. } \deg(u) < k\)
   - Output and remove \(u\)
2. \(\text{else}\)
   - \text{Contract a random edge}
One Iteration ($k = 3$)

while $G \neq$ empty
1. if $\exists u \in V$ s.t. $\deg(u) < k$
   • Output and remove $u$
2. else
   • Contract a random edge
**One Iteration** \((k = 3)\)

\[ k = 3 \]

\[
\begin{align*}
\text{while } & G \neq \text{empty} \\
1. & \text{ if } \exists u \in V \text{ s.t. } \deg(u) < k \\
& \quad \text{ Output and remove } u \\
2. & \text{ else} \\
& \quad \text{ Contract a random edge}
\end{align*}
\]
One Iteration ($k = 3$)

\[
\text{while } G \neq \text{ empty} \\
1. \quad \text{if } \exists u \in V \text{ s.t. } \deg(u) < k \\
   \quad \quad \bullet \text{ Output and remove } u \\
2. \quad \text{else} \\
   \quad \quad \bullet \text{ Contract a random edge}
\]
One Iteration ($k = 3$)

while $G \neq$ empty
1. if $\exists u \in V$ s.t. $\deg(u) < k$
   • Output and remove $u$
2. else
   • Contract a random edge
One Iteration ($k = 3$)

while $G$ ≠ empty

1. if $\exists u \in V$ s.t. $\deg(u) < k$
   - Output and remove $u$

2. else
   - Contract a random edge
One Iteration ($k = 3$)

Result
Removing a vertex with degree $< k$ ⇔ Separating by a cut with size $< k$

But, where should we contract?
→ We don’t know beforehand
→ We contract *randomly* (and repeat)
→ Is random really efficient? → **YES!** (good theoretical bounds)
When we successfully find a cut?

Or, when we fail to find the cut?

**Success**
When we contract all the vertices in one side before any cut edges

**Fail**
When we contract any cut edge before finding the cut
Random contraction has been a technique to design theoretical graph algorithms

- Min-cut [Karger, SODA’93]
- Steiner Cut, Node Multiway Cut [Chitnis+, FOCS’12]
- ...

Nice theoretical bound (our method also!)

The proposed method is the first empirically superior algorithm using random contraction
Previous vs. Proposed (MkECS algos)

**Previous** [Zhou+, EDBT’12]

**<Min-cut Algorithm>**
[Stoer, Wagner, ’97]

- 1 cut per 1 iteration
- Each iter. is slow

Simple application
Previous vs. Proposed (MkECS algos)

**Previous** [Zhou+, EDBT’12]

- **<Min-cut Algorithm>**
  - [Stoer, Wagner, ’97]

**Proposed**

- **<Min-cut Algorithm>**
  - [Karger, ’93]

- As a min-cut algorithm, slower than Stoer-Wagner

- Several substantial modifications

**Simple application**

- 1 cut per 1 iteration
- Each iter. is slow

- \( \geq 1 \text{ cuts} \) per 1 iteration
- Each iter. in **linear-time**
Outline

① High-level Description

② Implementation & Complexity

③ Number of iterations

One iteration

while $G \neq \text{empty}$

1. if $\exists u \in V$ s.t. $\deg u < k$
   • Output and remove $u$

2. else
   • Contract a random edge

for sufficient number of times

• Apply above algo. for subgraphs
Implementing an Iteration

Challenge:
Karger’s idea does not work
It does not really simulate random contraction, but just a binary search

Karger: Only contraction & interested only in final state

Proposed:
Contraction & Cut

Proposed:
At any moment
\[ \text{deg}(\nu) < k \]
Implementation

We really simulate random contraction
(in contrast to Karger’s min-cut)

How to process contraction efficiently?
• We maintain adjacency lists in hash dictionaries
• Contraction: merge two hash dictionaries
  – *Weighted Quick-find Algorithm* [Yao’76]

Time Complexity:
• Average time complexity $= O(|E|)$
Improving Technique: Forced Contraction

$k$ edges exist between two vertices
→ Immediately contract them

Since they will never be separated
Drastically decreases failure probability
while \( G \neq \text{empty} \)

1. if \( \exists u \in V \text{s.t.} \deg u < k \)
   - Output and remove \( u \)
2. else
   - Contract a random edge

for sufficient number of times

- Apply above algo. for subgraphs

Outline

1. High-level Description
2. Implementation & Complexity
3. Number of iterations
Necessary Number of Iterations

Separating $M_k$ECS $S$ with arbitrary high probability

*Easy bound:*

1. $O(|S|^2)$ iterations suffice [Karger’93]

*More involved analysis with forced contraction:*

2. $O(\log^2 |S|)$ iterations suffice

Completely decompose with small number of iterations

Theoretically guaranteed!
Experiments
Experiments: Number of iterations (1)

Iterations vs. Remaining cuts

| Dataset   | $|V|$   | $|E|$   | Type                  |
|-----------|--------|--------|-----------------------|
| Hollywood | 2.1 M  | 228 M  | Social network        |
| Indochina | 7.4 M  | 150 M  | Web graph             |
# Experiments: Number of iterations (2)

## Completely Decompose (100 runs)

| Dataset   | |V|   | |E|   | 種類               |
|-----------|---|-----|---|-----|-------------------|
| Hollywood | 2.1 M | 228 M | Social networks |
| Indochina | 7.4 M | 150 M | Web graphs     |
Experiments: Running time (small)

![Bar chart showing running times for different datasets and k values.](chart.png)

**Dataset** | **|V||** | **|E||** | **種類**
--- | --- | --- | --- | ---
**Arxiv-GrQc** | 5.2 K | 28 K | Social networks
**Epinions** | 76.8 K | 406 K | Social networks

Intel Xeon X5670 (2.93GHz), 48GB, C++(proposed), Java(previous)

7 × 10^4 times faster!
Experiments: Running time (large)

Can handle networks with **hundreds of millions** of edges

| Dataset      | $|V|$  | $|E|$  | Type            |
|--------------|------|-------|-----------------|
| India        | 1.4 M| 17 M  | Web graphs      |
| LiveJournal  | 4.8 M| 69 M  | Social networks |
| Indochina    | 7.4 M| 150 M | Web graphs      |
| Hollywood    | 2.1 M| 228 M | Social networks |

Intel Xeon X5670 (2.93GHz), 48GB, C++
Conclusion

Finding cohesive groups in graphs

- Serious problem of classic models
- Recent model: $M_kECS$
- Well-connected, but the enumeration algorithm was slow

New algorithm for enumerating $M_kECS$s

- Based on random contraction
- Works in nearly linear-time (with theoretical guarantee)
- Several orders of magnitude faster than previous algo.

Software available: [http://git.io/mkecs](http://git.io/mkecs)